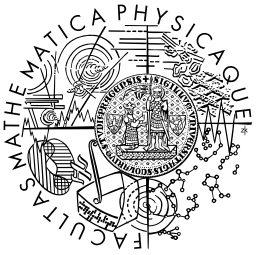


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Martin Šerý

Analýza videa pro entomologii

Katedra software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán
Studijní program: Informatika
Studijní obor: IOI

Praha 2017

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Českých Budějovicích dne ..... Podpis autora

Název práce: Analýza videa pro entomologii

Autor: Martin Šerý

Katedra: Katedra software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Katedra software a výuky informatiky

Abstrakt: Analýza vzorců chování jedinců druhu *Drosophila melanogaster* je předmětem studia mnoha laboratořích. Sleduje se chování jedinců například v různých genových liniích. K tomu, aby byla tato analýza možná, je potřeba mít zařízení, které je schopné mušky sledovat a poskytnout o nich strojově čitelné informace. Tyto údaje jsou výstupem vyvinutého trasovacího programu `FlyCatcher`, který zaznamenává pozice sledovaných objektů. Díky tomu můžeme určit polohu, rychlost vzájemné postavení jedinců a další parametry. `FlyCatcher` je postaven na využití knihovny `AForge`, která slouží ke zpracování multimediálních souborů. `FlyCatcher` umožňuje zpracování videí pořízených v laboratoři a poskytnutí výstupů ve formátu `.csv`. Tento formát je nadále zpracováván externími statistickými programy.

Klíčová slova: analýza videa, Drosophila, sledování objektů

Title: Video analysis for entomology

Author: Martin Šerý

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: Analysis of behavioral patterns of *Drosophila Melanogaster* species is a subject of study of many laboratories. The behavior of individuals, for example in different gene lines, is monitored. In order for this analysis to be possible, it is necessary to have a device that is capable of tracking the fly and providing machine-readable data about them. These data are the output of a trace program `FlyCatcher`, which records the positions of the observed objects. Thanks to this, we can determine the location, the mutual position of individuals and other parameters. `FlyCatcher` is based on the use of the `AForge` library which handles multimedia files. `FlyCatcher` lets us process videos made in the lab and provide outputs in the `.csv` format. This format is further processed by external statistical programs.

Keywords: video analysis, Drosophila, object tracking

Chtěl bych poděkovat svému vedoucímu RNDr. Pelikánovi za rady & pomoc při vypracování této práce. Dále pak pracovníkům z Laboratoře molekulární genetiky, Entomologický ústav, Biologické centrum AV ČR za námět na tuto práci. V neposlední řadě pak svým rodičům za pomoc s korekturami & kontrolou typografie.

Obsah

Úvod

Laboratoř molekulární genetiky, Entomologický ústav, Biologické centrum AV ČR se (mimo jiné) zabývá studiem octomilek (*Drosophila melanogaster*). Součástí tohoto výzkumu je také analýza vzorců chování much v různých genových liniích. A právě k tomu, aby byla tato analýza možná, je potřeba mít systém, který je schopen mušky sledovat a poskytnout o nich strojově čitelné informace.

Aktuálně jsou v Laboratoři molekulární genetiky používány programy, jejichž použití je uživatelsky náročné a nebo jsou nevhodné pro dané testování. Z tohoto důvodu vznikl návrh na vytvoření programu `FlyCatcher`, který by měl tyto nedostatky odstranit.

Finální verze vytvořeného programu je naprogramována v jazyce `C#` pro prostředí `.NET`. Tato varianta byla zvolena na základě dobrých zkušeností s udržitelností kódu v jazyce `C#`. Aplikace byla postavena na využití knihovny `AForge.NET`, která umožňuje práci s videem a zpracování obrazu.

1. Úvod do problematiky

Jedním z živočišných druhů, kterým se Laboratoř molekulární genetiky na Entomologickém ústavu Biologického centra AV ČR zabývá, je octomilka obecná (*Drosophila melanogaster*). Součástí tohoto výzkumu je i sledování chování těchto much v různých podmínkách (viz Zhang a kol., 2010), jako je třeba nárůst stresu v případě nedostatku spánku, frekvence páření při mutaci zkoumaného genu a nebo aktivita jedinců v různých okolních podmínkách (Colomb a kol. (2012), Donelson a kol. (2012)). Nadále budeme termínem „objekty“ nazývat sledované jedince.

1.1 Definice problému

Právě analýza projevů, které jsou vizuálního charakteru (páření, pohyb...), jsou předmětem zájmu této práce. Objekty jsou umístěny do sledovací arény, která může být osvětlena buď zespoda a nebo zeshora. Kamera, ze které je pořizován záznam, je umístěna nad arénou. Tento záznam je poté potřeba zpracovat. Zpracování videa je definováno jako poskytnutí faktických informací o daném pozorování ve strojově zpracovatelné podobě — v našem případě `.csv` formát. Pozorovanými informacemi jsou primárně pozice jednotlivých objektů na každém snímku, orientace v prostoru, rychlost a nebo velikost jedinců (např. rozlišení pohlaví).

Všechna videa jsou pořizována v podmínkách, které umožňují co nejsnažší a deterministickou selekci jedinců — tj. dostatečný kontrast mezi jedinci a podkladem. Veškeré trasování probíhá ve dvou dimenzionálním prostoru.

1.2 Struktura práce

V následující kapitole je provedena diskuze řešení, kde jsou rozebrány problémy s jednotlivými přístupy k řešení problému. Dále je zde nastíněn finální návrh programu.

Poté následuje návrh aplikace z programátorského hlediska. Z důvodu zpětného odkazovnání je nejprve obecně popsána knihovna `AForge`. Následně jsou popsány konkrétní třídy a metody z této knihovny, které byly použity k implementaci programu `FlyCatcher`. K pochopení samotné dokumentace je důležité vědět co použité třídy dělají.

Ve dvou následujících kapitolách se poté nachází podrobná programátorská dokumentace. V kapitole o `API` je popsána vnitřní struktura aplikace, která zajišťuje požadovanou modularitu. Následně je podrobně popsána samotná implementace připraveného `API`.

Součástí této práce je i návod, ve kterém je znovu a jednodušeji popsáno chování a ovládání programu `FlyCatcher`. Tato kapitola by měla sloužit jako uživatelský manuál k celé aplikaci.

V poslední kapitole se nachází finální zhodnocení celé práce.

2. Diskuze řešení

Zadanou problematikou se zabývala diplomová práce, kterou vypracoval Štefan Kakaš (2012), součástí jejíž realizace bylo sestavení `HW` aparatury a napsání trasovacího programu `Drosana`. Aparatura je stále aktivně využívána, ovšem `SW` `Drosana` má problémy s přechodem na `OS` `Windows 10`. Další nevýhody spočívají v samotném návrhu aplikace. Jedná se například o obtížnou rozšiřovatelnost funkcí programu, nebo aktviní využívání systémové schránky ke zpracování videa, čímž je omezena další práce na počítači v případě spuštění analýzy videa v programu `Drosana`.

Z tohoto důvodu mi byla navržena spolupráce s Laboratoří molekulární genetiky za účelem vytvoření nového trasovacího programu, který by měl být uživatelsky příjemnější než `Drosana` a který by výhledově umožnil rozšířit možnosti analýzy chování.

2.1 Cíle projektu

Nový program musí být schopen ze zadaného videa v běžném formátu (`.avi`, `.mpeg`, `.mp4`...) extrahovat požadované informace v anotované tabulce. Těmito informacemi jsou:

- Pozice jednotlivých objektů,
- Velikost objektů,
- Rychlost pohybu objektů.

V laboratoři se většinou pokusy opakují několikrát za sebou. Proto je velmi užitečné umět ukládat již nastavené parametry pokusu, aby odpadla nutnost je nastavovat při každé iteraci znovu.

Dále by bylo vhodné, aby byl výsledný program navržen tak, aby umožňoval snadnou rozšiřitelnost o nové funkce. Takovýmto rozšířením programu může být detekce a zachycení specifických situací jako je například páření či trhavé pohyby.

2.2 Návrh řešení

Při návrhu řešení této aplikace, byla použita externí knihovna `AForge`. Žádné další externí knihovny použity nebyly. Zdůvodnění výběru této knihovny se nachází níže.

2.2.1 Zpracování videa

Důležitou složkou aplikace je možnost zpracovávat videa. Videa lze zpracovávat přímo jako binární soubory, ale to je zdlouhavé a velmi náchylné na chyby ze strany programátora. Další možností by bylo použít některé externí programy, které by byly schopné z videa vytvářet snímky a tyto poté předávat aplikaci. Tím by ovšem vznikala velká závislost na této externí aplikaci a výsledný program by nebyl tak modulární.

Proto byla zvolena varianta již předpřipravených knihoven. V prvotních fázích se rozhodovalo mezi `OpenCV` a `AForge`. Primární kritérium výběru byla kombatibilita s jazykem `C#` a s prostředím `.NET`. Nakonec byla zvolena knihovna `AForge`, protože je napsaná přímo v jazyce `C#` pro prostředí `.NET`. Dále pak nabízí další funkce, které byly pro správnou funkci aplikace potřeba (viz kapitola 4).

2.2.2 Zpracování jednotlivých snímků

Na snímcích, které jsme získali ze zpracovávaného videa, je potřeba nalézt zájmové objekty a tyto poté trasovat. Výchozí možnosti jazyka `C#` co se týče zpracování obrázků jsou omezené a dost často pomalé. Proto bylo opět přistoupeno k variantě externích knihoven. Nejprve byla vyzkoušena knihovna `AForge`.

Bylo zjištěno, že vybraná knihovna poskytuje veškerou další potřebnou funkcionalitu na zpracování obrazu. Díky tomu je struktura aplikace přizpůsobena vybraným třídám z knihovny `AForge`.

2.2.3 Trasování

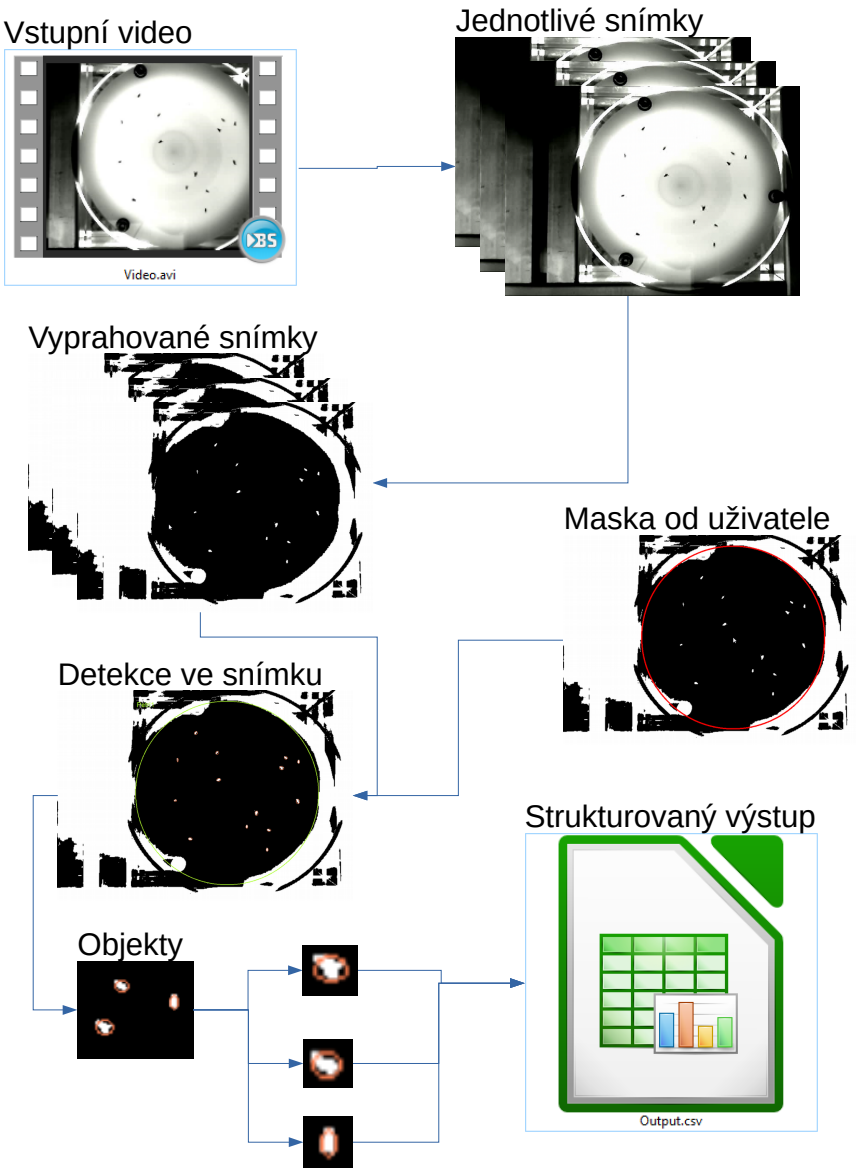
Největším problémem řešení bylo samotné trasování objektů – konkrétně mapování objektů na jejich trasy. K řešení tohoto problému byla zvolena varianta, kdy hledáme perfektní párování minimální váhy na úplném bipartitním grafu, kde jedna partita označuje zájmové objekty a druhá pak trasy. Ohodnocení hran v grafu odpovídá míře podobnosti trasy a zájmového objektu.

K řešení tohoto problému byl díky své snadné implementaci zvolen algoritmus popsaný v Kuhn (1955), který byl následně modifikován do takové podoby, aby vyhovoval našemu zadání (viz 6.8).

2.3 Struktura programu

Výsledný program ve zpracovávaném videu nejprve lokalizuje zájmové objekty, které zpracuje, a následně vypíše do `.csv` tabulky zjištěné informace.

Proces zpracování zájmových objektů je popsán na obrázku 2.1. Ve finálním programu představuje každý krok na tomto diagramu jedno z připravených rozhraní. Programátorská struktura celé aplikace včetně použitých rozhraní se nachází na začátku kapitoly 6.



Obrázek 2.1: *Diagram běhu programu*

3. O knihovně AForge.NET

Tato kapitola slouží k seznámení čtenáře s knihovnou AForge, která je důležitou součástí výsledného programu. Jedná se o open source knihovnu, určenou ke zpracování zdrojů v oblasti multimediální a umělé inteligence. Knihovna zahrnuje třídy na zpracování obrazu či videa, genetické algoritmy, algoritmy pro strojové učení a nebo neuronové sítě.

Celý framework je implementovaný pro jazyk C#, což umožňuje velmi snadné implementování aplikací pro platformu Windows.

Důvodem výběru této knihovny byla snadná dostupnost a šíře záběru celé knihovny. Díky tomu je výsledné aplikace závislá pouze na jedné externí knihovně. S vybranou knihovnou nebyly, díky podrobné online dokumentaci, žádné problémy, které by nešli rychle vyřešit.

3.1 Instalace & použití

Aktuální verze této knihovny je ke stažení na adrese www.aforgenet.com. Samotné použití je díky velkému množství jednoduchých vzorových příkladů jednoduché. Mezi takovéto příklady patří na příklad jednoduché přehrávače videa, či základní detektor pohybu.

3.2 Projekty využívající AForge.NET

Knihovna AForge má velmi široké využití v různých praktických aplikacích. Toto potvrzuje i seznam projektů, které využívají právě knihovnu AForge:

- *Cats Eyes Alarm Trigger* - více kamerový detektor pohybu,
- *Universe Sandbox* - interaktivní astronomický software,
- *iSpy* - detektro pohybu a zvuku s živým přenosem,
- *Kinovea* - analyzátor videa pro trenéry, sportovce a zdravotníky,
- *Motion-activated TimeLapse Video Recorder* - pohybem aktivovaný nahrávač videa.

4. Použité třídy z knihovny AForge.NET

Tato kapitola se zabývá představením tříd z knihovny `AForge`, které jsou využívány v programu `FlyCatcher`. Pro pochopení kapitol, které se zabývají samotnou programátorskou dokumentací, je potřeba alespoň základní znalost těchto tříd.

V průběhu implementace bylo použito několik tříd z knihovny `AForge.NET`. Jednalo se o využití tříd, které umožňují přehrávání videa — `VideoFileReader` a `AVIReader`. Ostatní třídy se nacházejí ve jmenném prostoru `Imaging`, kde se nacházejí právě třídy pro zpracování obrázků. Jedná se o třídu `Blob`, která slouží pouze k uchování informací o shluku pixelů v daném obrázku. Nebo `BlobCounter`, který je použit na detekci zájmových jedinců. Další použité třídy jsou filtry, které alternují vstupní snímek.

4.1 BlobCounter

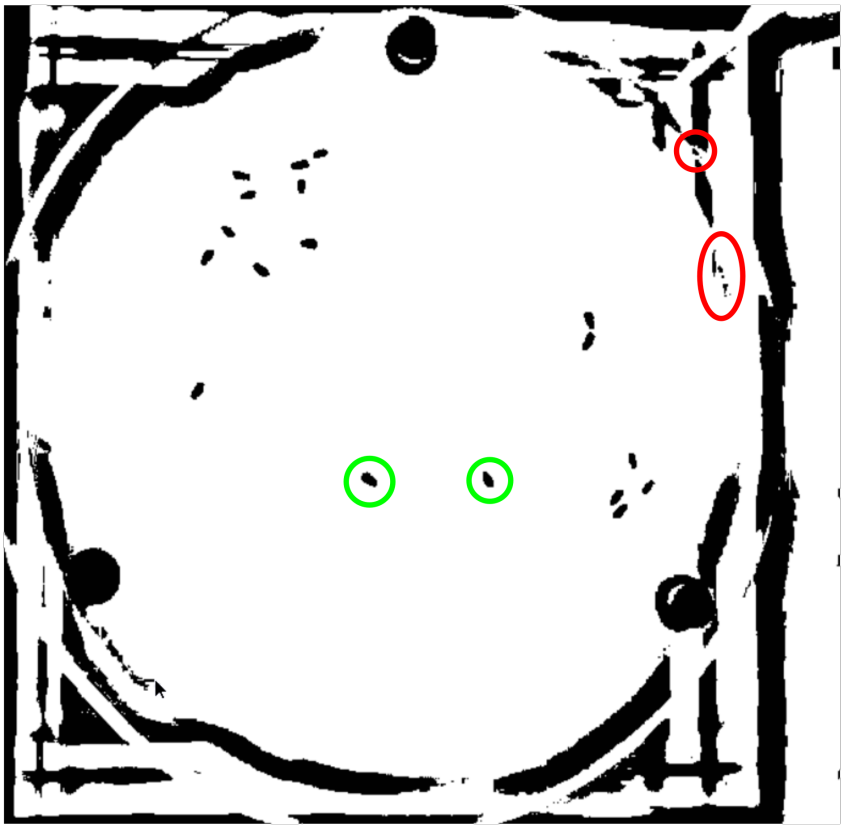
Tato třída slouží k detekování shluků pixelů v obrázku ve stupních šedi. Díky tomuto formátu vstupního snímku je možné reprezentovat odstín šedi jako jedno přirozené číslo z rozsahu $[0, 255]$. Jednotilvé instance jsou inicializovány buď obrázkem nebo ničím — to v případě, že obrázek bude dodán později a nebo se jedna instance s jedním nastavením bude používat na zpracování více obrázků, jako v našem případě.

V obecném případě má definovanou prahovou hodnotu, podle které rozděluje pixely na `pozadí` a `zájmové shluky`. Kde pozadí je to co je pod prahem a zájmové shluky jsou nad prahem. Jak ukazuje obrázek 4.1 (pro lepší kontrast a viditelnost byly barvy v obrázku invertovány) jsou tyto shluky později rozděleny na reálné objekty zájmu (zelená) a šum (červená).

Samotné využití této třídy spočívá v metodě `ProcessImage`, která přijímá jako argument obrázek a inicializuje tak instanci `BlobCounter` do stavu, kdy je možné zavolat metodu `GetObjectsInformation`, která vrací jako výstup pole nalezených shluků ve formě instancí třídy `Blob`.

Tyto shluky lze již filtrovat (viz obr. 4.1) na šum, který je zahozen, a na objekty zájmu. K tomuto účelu slouží přepínač `FilterBlobs`, který slouží k jednoduchému filtrování shluků. `FilterBlobs` nastavuje, zda se vůbec mají shluky filtrovat nebo ne. Základní filtrování probíhá pouze podle velikosti (šířka a výška) shluku. Tyto mezní hodnoty si `BlobCounter` udržuje v proměnných `MaxHeight` pro maximální výšku, `MinHeight` pro minimální výšku a analogicky `MaxWidth` a `MinWidth` pro šířku.

Další přepínač je `CoupledSizeFiltering`, který mění přístup k minimální a maximální velikosti vydaných instancí třídy `Blob`. Výchozí nastavení je, že jakmile je překročena jakákoliv hodnota z mezních hodnot pro velikost shluku, tak je shluk odfiltrován. Pokud ovšem nastavíme `CoupledSizeFiltering` na `true`, tak poté je potřeba překročit limity jak v x-ovém směru tak v y-ovém. To v našem případě umožní lepší filtrování mušek, které jsou protažené v podélné ose těla. Tím pádem filtrujeme pouze podle největšího naměřeného rozměru.



Obrázek 4.1: *Rozdíl mezi objektem zájmu & šumem*

V případě potřeby je možné dodat i specializovaný filtr shluků. Stačí vytvořit třídu, která bude odvozena od rozhraní `IBlobsFilter`, které vyžaduje právě jednu metodu. Touto metodou je predikát `Check`, který přijímá jako jediný argument instanci třídy `Blob`. Jejím výstupem je booleovská hodnota, která určuje, jestli shluk má, nebo naopak nemá být zahrnut ve finálním výstupu (`true` znamená, že `Blob` bude zahrnut). Instance vytvořeného filtru je poté dosazena do vlastnosti `BlobsFilter`, která se nachází na instanci třídy `BlobCounter`.

Takto by vypadalo využití této třídy v praxi:

```
1  BlobCounter bc = new BlobCounter();
2  bc.FilterBlobs = true;
3  //Minimální velikost shluku je 5 na 5 pixelů
4  bc.MinWidth = 5;
5  bc.MinHeight = 5;
6  bc.ProcessImage(image);

7  foreach (Blob blob in bc.GetObjectsInformation(image, false))
8  {
9      //Zpracování shluků
10 }
```

4.2 Blob

Třída `Blob` slouží k uchovávání informací o shluku pixelů. Jedná se o výstup metody `GetObjectsInformation` zavolanou nad třídou `BlobCounter`. Uchovávané informace zahrnují oblast, kterou shluk zabírá v původním obrázku. Tato oblast je reprezentována instancí třídy `Rectangle`, která se nachází ve jmenném prostoru `System.Drawing`.

Další informace je počet pixelů ve shluku (`Area`), která je interně reprezentována jako běžné celé číslo. Poslední použitý údaj je těžiště (`CenterOfGravity`), reprezentované jako instance třídy `Point` ze jmenného prostoru `AForge`. Tento dvoudimenzionální bod má oproti bodu ze jmenného prostoru `System.Drawing` doimplementováno několik funkcí, jako je třeba počítání euklidovských vzdáleností, či normalizace vektoru.

4.3 Filtry

Filtry se používají k cílené změně vstupního obrázku. V našem případě byly použity tři druhy filtrů. Algoritmicky nejjednodušší je filtr, který invertuje v obrázku všechny barvy. Ten je použit v případě, že si sám uživatel zaškrtne jeho použití, a hodí se pokud chceme sledovat tmavé objekty na světlém pozadí.

Další filtr je převod na stupně šedi. Na tento převod je ve jmenném prostoru `AForge.Imaging.Filters` napsaná knihovna `Grayscale`, která počítá výslednou barvu pixelu jako vážený průměr s koeficienty pro jednotlivé barevné složky. Jednotlivé koeficienty (pro barevné kanály červená, zelená a modrá) si nastavuje sám uživatel. Výchozí nastavení poměrů je zvoleno stejné jako v běžných *grayscale* algoritmech (viz Poynton, 1998).

```
red_coef: 0.2125
green_coef: 0.7154
blue_coef: 0.0721
```

Což je stejné nastavení parametrů jako v algoritmu `BT 709`, který je implementován ve třídě `Grayscale.CommonAlgorithms`, a zachovává stejnou jasovou složku pixelů jako v původním barevném obrázku. Výsledná barva se poté spočítá jako vážený průměr `RGB` barvy původního pixelu.

```
gray = (red_coef * red + green_coef * green + blue_coef * blue)
```

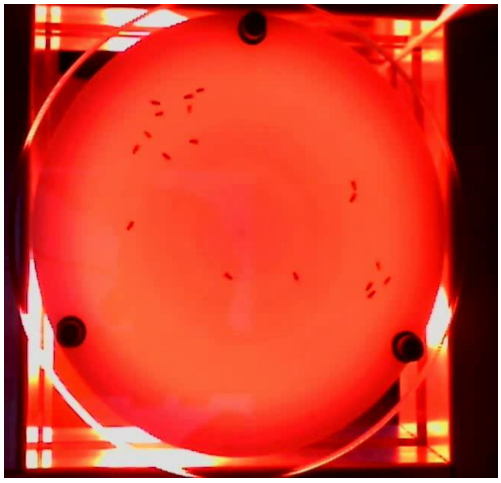
Převod na stupně šedi je důležitý kvůli poslednímu filtru, který přijímá pouze obrázky s jedním barevným kanálem. Tento filtr přijme obrázek a na výstup vydá vyprahovaný obrázek — tj. pouze dvě barvy černá nebo bílá. K prahování je v knihovně `AForge` připraveno několik algoritmů.

Prvním je obyčejný `Threshold`, kterému se nastaví prahová hodnota jasu, a vše co je nad ní je považováno za černou, co je pod touto hodnotou je bílá. Právě kvůli tomuto porovnání je potřeba mít obrázek ve stupních šedi, aby měl každý pixel právě jednu barevnou hodnotu. Výchozí hodnota prahové hodnoty je 128.

Další algoritmy pracují s počítáním prahové barvy na základě analýzy snímku, ale poté spustí prahování stejně jako `Threshold`. Jedná se o třídu `OtsuThreshold`,

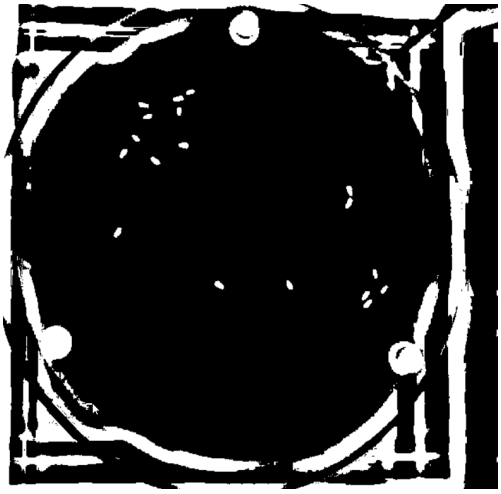
která implementuje algoritmus popsaný v práci Otsu (1979). Následující třída `SISThreshold` počítá výslednou prahovou hodnotu analýzou gradientů jednotlivých pixelů. Poslední třída `IterativeThreshold` iterativně vylepšuje výchozí prahovou hodnotu (viz Gonzalez, 2009, kap. 10).

Nejlepších výsledků (obr 4.2) v prahování much ovšem bylo docíleno s pomocí třídy `BradleyLocalThresholding`. Tato třída implementuje algoritmus popsaný v Bradley a Roth (2007). Nevýhodou použití této třídy je zvýšení časové náročnosti na zpracování jednoho snímku. Proto byla zvolena varianta, kde uživatel vybírá algoritmus pro prahování.



(a) Originál

(b) OTSU



(c) Bradley

Obrázek 4.2: *Porovnání tří prahovacích funkcí*

4.4 Přehrávače videa

Třídy `AVIReader` a `VideoFileReader` slouží k otevření videa a jeho následnému zpracování po snímcích. Tyto třídy podpurují pouze videa v některých formátech `.avi` respektive `.mpeg`. Tuto restrikci musí respektovat i celý program.

V knihovně se dále nachází rozhraní `IVideoSource`, které definuje přehrávače videa. Bohužel toto rozhraní neumožňuje ve videu vyhledávat a nebo i pozastavit

přehrávání bez nutnosti spouštět následně celé přehrávání od začátku. Díky tomu nebylo toto rozhraní použito, přestože by jeho použití umožnilo zpracovávat více formátů.

```
interface IVideoSource
{
    long BytesReceived { get; }
    int FramesReceived { get; }
    bool IsRunning { get; }
    string Source { get; }

    event NewFrameEventHandler NewFrame;
    event PlayingFinishedEventHandler PlayingFinished;
    event VideoSourceErrorEventHandler VideoSourceError;

    void SignalToStop();
    void Start();
    void WaitForStop();
}
```

5. API programu FlyCatcher

Předmětem této kapitoly je představení rozhraní, která vznikla v rámci přípravy komplexní aplikace. Dohromady bylo vytvořeno několik modulárních tříd (viz příloha Zdrojový kód), které na sebe nemají pevné vazby, pouze splňují minimální rozhraní potřebné pro jejich interakci, která je popsána na začátku kapitoly 6. Celkem bylo vytvořeno 6 hlavních rozhraní, jejichž rozbor je předmětem této kapitoly.

5.1 IGiver

Jak již název napovídá, toto generické rozhraní se používá jako podavač jednotlivých objektů. Jediný typový parametr rozhraní určuje, jaké objekty budou danou instancí, implemetující toto rozhraní, podávány.

```
1  interface IGiver<out T> : IEnumerable<T>
2  {
3      T First { get; }
4      T Actual { get; }
5      T Last { get; }
6      T this[int index] { get; }

7      bool isValidIndex(int index);
8      int actualIndex { get; set; }

9      int RunFrom { get; set; }
10     int RunTo { get; set; }

11     int RunFromMin { get; }
12     int RunToMax { get; }
13     int Step { get; set; }

14     string Path { get; }
15     string Tag { get; }
16 }
```

Součástí rozhraní jsou přístupové metody k objektům, dále pak klasická číselná indexace a práce se samotnými indexy:

- `First` - vrátí objekt na počátku enumerace,
- `Actual` - vrátí prvek na aktuálním indexu,
- `Last` - vrátí poslední prvek enumerace,
- `this[int index]` - vrátí prvek s indexem `index`,
- `isValidIndex(int index)` - zkontroluje jestli je zadaný `index` validní,
- `actualIndex` - vrátí či nastaví aktuální pozici v enumeraci,
- `RunFrom` - od kterého indexu probíhá enumerace (nativně 0),
- `RunTo` - do kterého indexu probíhá enumerace (nativne `RunToMax`),

◁ `RunFromMin` - nejmenší validní index,
◁ `RunToMax` - nejvyšší validní index,
◁ `Step` - změna indexu mezi snímky,
◁ `Path` - plná cesta ke zpracovávanému souboru,
◁ `Tag` - název zpracovávaného souboru.

5.2 IPreProcessor

`IPreProcessor` definuje generické rozhraní, jehož jedinou funkcí je přetransformovat vstupní instanci na jinou instanci téže třídy.

```
1 interface IPicturePreProcessor<T>
2 {
3     T processImage(T image);
4 }
```

Součástí rozhraní je jediná metoda `processItem`, která jako parametr přijímá objekt k transformaci a na výstup vydá výsledek transformace.

◁ `processItem(T item)` - přetransformuje `item`.

5.3 IMask

`IMask` definuje jednoduché generické rozhraní s typovým parametrem o kterém rozhoduje, jestli je a nebo není validní.

```
1 interface IMask<T>
2 {
3     string tag { get; }
4     bool isIn(T point);
5     string PrintOut();
6     void drawMask(Graphics gr);
7 }
```

Instance implementující toto rozhraní mají svůj (ideálně) jedinečný `tag`, umí samy sebe nakreslit v zadané instanci třídy `Graphics` a dokáží posoudit, jestli je zadaná instance třídy `T` předmětem zájmu či ne.

◁ `Tag` - textová reprezentace „jména“ instance,
◁ `IsIn(T point)` - určí jestli je `point` zájmový či nikoliv,
◁ `PrintOut()` - vrací textový popis instance,
◁ `DrawMask(Graphics gr)` - slouží ke grafickému znázornění této instance.

5.4 ICounter

Rozhraní `ICounter` slouží k extrakci zájmových objektů.

```
interface ICounter<in T1, T2, T3>
{
    IEnumerable<Tuple<string, T2>> CountItems(T1 image);
    ICollection<IMask<T3>> Masks { get; set; }
}
```

Toto rozhraní poskytuje ven kolekci masek (tj. instance rozhraní `IMask<T3>`) a metodu na extrakci objektů `T2`, které se získají z instance třídy `T1`.

- `Masks` - výsledné objekty se párují s označením masky, podle toho které splní podmínku `IsIn`,
- `CountItems(T1 image)` - vrací enumeraci dvojic `označení masky, zájmový objekt`.

5.5 IData

V instancích implementujících toto rozhraní jsou uložena trasovací data o objektu.

V rozhraní `IData` jsou uložena veškerá data o trasách jednotlivých objektů. Dále poskytuje metody na porovnávání objektů s trasou, či predikce chování.

- `Items` - objekty, které si pamatujeme, abychom byly schopni provádět predikce,
- `averageSpeed` - průměrná změna vzdáleností dvou jednotlivých objektů,
- `averageArea` - průměrná velikost objektu,
- `immediateSpeed` - průměrná změna vzdáleností dvou jednotlivých objektů,
- `immediateArea` - průměrná velikost objektu,
- `Direction` - směr pohybu sledovaného objektu,
- `PredictNext` - predikce pozice výskytu objektu v další iteraci,
- `First` - poslední přidaný objekt,
- `Tag` - označení trasy,
- `ValidValue` - určuje míru invalidace trasy,
- `Valid` - proměnná určující, zda je daná trasa validní či nikoliv,
- `MakeInvalid()` - metoda, která zvýší míru invalidace,
- `Revalidate()` - metoda, která sníží míru invalidace,
- `GetMatch(T item)` - vrací míru příslušnosti objektu `item` k trase,
- `AddItem(T item)` - přiřadí objekt `item` k trase,
- `PrintOut(StreamWriter writer, OutputFormat format)` - vypíše aktuální stav trasy dle zadaného formátu,
- `PrintOutHeader(StreamWriter writer, OutputFormat format)` - vypíše hlavičku dle zadaného formátu,
- `PrintOutTag(StreamWriter writer, OutputFormat format)` - vypíše označení trasy dle zadaného formátu,
- `Draw(Grpahics gr, HighlightFormat format)` - vykreslí aktuální stav trasy dle zadaného formátu.

```
1   interface IData<T, out MeasureType, out DistanceType, out
    ↪ PositionType>
2   where MeasureType : IComparable
3   {
4       ICollection<T> Items { get; }

5       DistanceType averageSpeed { get; }
6       DistanceType averageArea { get; }
7       DistanceType immediateSpeed { get; }
8       DistanceType immediateArea { get; }
9       PositionType Direction { get; }
10      PositionType PredictNext { get; }

11      T First { get; }

12      string Tag { get; }

13      int ValidValue { get; }
14      bool Valid { get; }
15      void MakeInvalid();
16      void Revalidate();

17      MeasureType GetMatch(T item);
18      MeasureType AddItem(T item);

19      void PrintOut(StreamWriter writer, OutputFormat format);
20      void PrintOutHeader(StreamWriter writer, OutputFormat
        ↪ format);
21      void PrintOutTag(StreamWriter writer, OutputFormat
        ↪ format);
22      void Draw(Graphics gr, HighlightFormat format);
23  }
```

5.6 IKeeper

Instance rozhraní `IKeeper` slouží k uchovávání všech nalezených tras a hromadnému přiřazování objektů k jednotlivým trasám.

`IKeeper` udržuje v jedné kolekci `ItemsData` nalezené trasy, s jejichž daty se manipuluje přes metody `ActualizeData`, `Refresh` a `Init`.

```
interface IKeeper<ItemType, MeasureType, DistanceType,
 ↪  PositionType>
    where MeasureType : IComparable
    where DistanceType : IComparable
{
    ICollection<IData<ItemType, MeasureType, DistanceType,
     ↪  PositionType>> ItemsData { get; }

    string Tag { get; }

    void ActualizeData(IEnumerable<ItemType> items);
    void Refresh(IEnumerable<ItemType> items);

    void PrintOut(StreamWriter writer, OutputFormat format);
    void PrintOutHeader(StreamWriter writer, OutputFormat
     ↪  format);
    void PrintOutTag(StreamWriter writer, OutputFormat format);
    void Draw(Graphics gr, HighlightFormat format);
}
```

- `ItemsData` - kolekce tras,
- `Tag` - označení instance,
- `ActualizeData(IEnumerable<ItemType> items)` - metoda, která přidá nové objekty do příslušných tras,
- `Refresh(IEnumerable<ItemType> items)` - inicializuje každou uchovanou trasu do výchozího stavu,
- `PrintOut(StreamWriter writer, OutputFormat format)` - vypíše aktuální stav tras dle zadaného formátu,
- `PrintOutHeader(StreamWriter writer, OutputFormat format)` - - vypíše hlavičky dle zadaného formátu,
- `PrintOutTag(StreamWriter writer, OutputFormat format)` - vypíše označení tras dle zadaného formátu,
- `Draw(Grpahics gr, HighlightFormat format)` - vykreslí aktuální stav tras dle zadaného formátu.

6. Implementace API

V předchozích částech bylo nastíněno s jakými rozhraními a s jakými datovými strukturami z knihovny AForge budeme pracovat. Každá implementace představených rozhraní je vyměnitelná za jinou, která může změnit výslednou funkci celé aplikace.

6.1 Provázání rozhraní

Tato sekce se zaměří na samotný popis použití popsaných tříd uvnitř programu FlyCatcher.

Vstupní video soubor je předán instanci implementující rozhraní IGiver, tato instance slouží k rozdělení vstupu na jednotlivé snímky. Poté jsou jednotlivé snímky zpracovány třídou implementující IPreProcessor, který připraví snímky k dalšímu zpracování. Instance rozhraní ICounter dostane připravenou masku IMask od uživatele a provede detekci objektů ve snímku. Nalezené objekty jsou poté předány instanci IKeeper, kde dojde k rozdělení jednotlivých objektů k příslušným IData strukturám. Tyto pak poskytují uživateli strojově zpracovatelný výstup. Celý tento proces je znázorněn na obrázku 6.1.

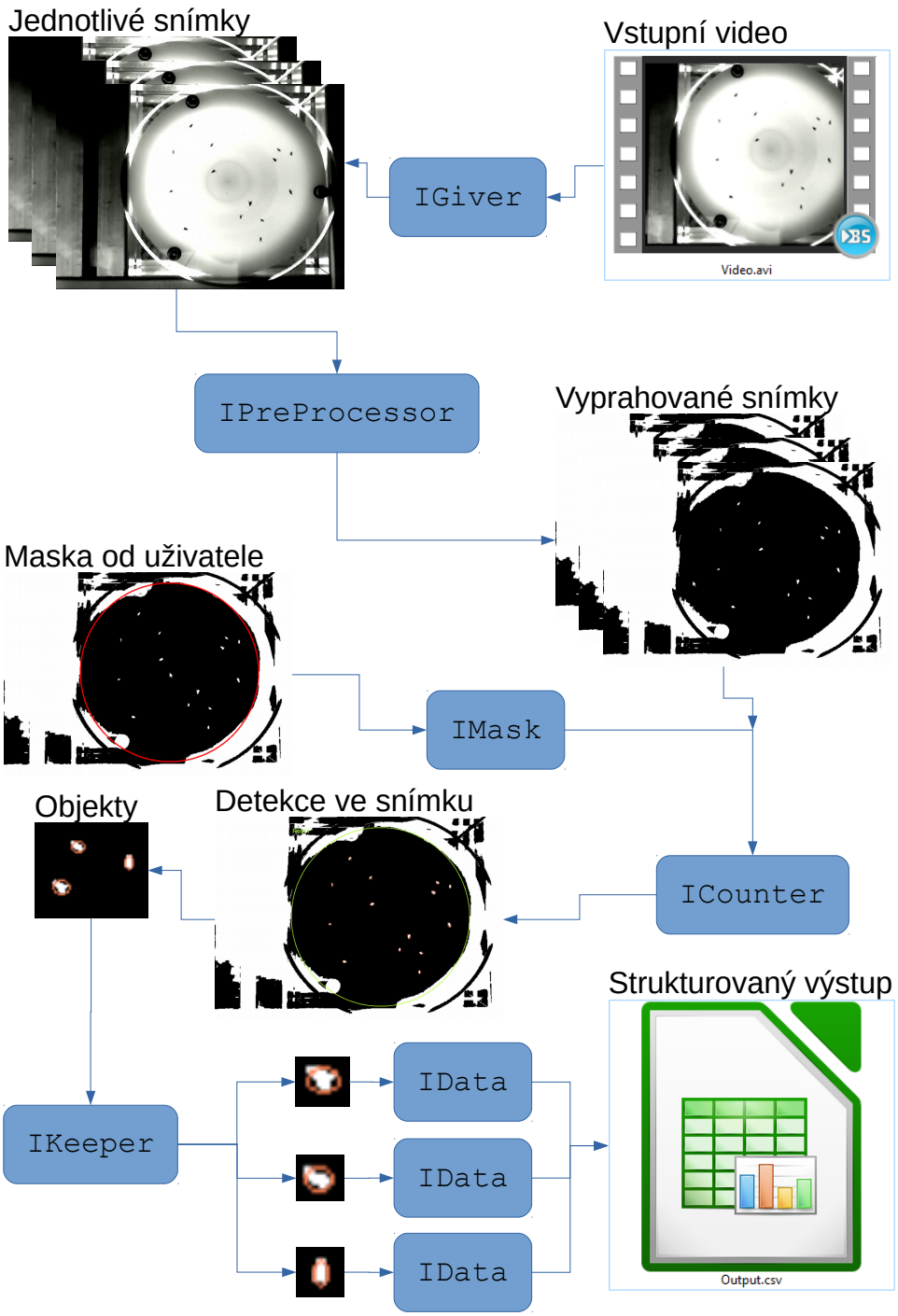
6.2 SeparatePhotoGiver

Tato třída slouží ke zpracování velkého množství očíslovaných obrázků. Třída byla implementována pouze jako testovací a v budoucnu by měla být rozšířena na zpracování videií v reálném pořizovacím čase. To by mělo zajistit Raspberry Pi s kamerou. Knihovna AForge nabízí třídu VideoCaptureDevice:IVideoSource, která umožňuje zpracovávat video pořízené z HW kamer na zařízení – webkamery apod.

```
class SeparatePhotoGiver : IGiver<Bitmap>
```

Implementuje rozhraní IGiver<Bitmap>, díky čemuž je patrné, že se jedná o podavač obrázků v bitmapové formě. SeparatePhotoGiver dostává jako parametr konstruktoru cestu k fotografiím. V enumeraci poté bere očíslované fotografie od nejmenší po největší.

SeparatePhotoGiver předpokládá, že v parametru konstruktoru path je cesta k poslednímu snímku, který chceme zpracovávat. Pořadové číslo tohoto snímku totiž určuje horní zarážku na zpracování. Další předpoklad je, že zpracujeme všechny snímky od 0 až do snímku daného na vstupu. Tedy ve chvíli předání souboru Mouchy50.jpg začne analýza od snímku Mouchy00.jpg, následuje snímek Mouchy01.jpg a tak dále až ke snímku Mouchy50.jpg. Toto rozložení může ovlivnit parametr step.



Obrázek 6.1: *Diagram zpracování multimediálního vstupu*

6.3 VideoMPEGGiver & VideoAVIGiver

Vhodným nahrazením třídy `SeparatePhotoGiver` v programu je třída, schopná zpracovávat celá videa a sama je rozdělit na jednotlivé snímky. Touto třídou jsou právě `VideoMPEGGiver` a `VideoAVIGiver`, které zpracovávají videa ve formátu `.mpeg` a nebo `.avi`.

Nevýhodou `.mpeg` videí je nedostatečná podpora zpracování těcho videí v knihovně `AForge`. Díky tomu jsou v případě `.mpeg` videí omezeny funkce pro prohlížení videa.

Problémem s přehráváním `.avi` videí, může být absence kodeků v systému. Na stránkách `AForge`, se píše, že třída `AVIReader` staví nad zastaralým `API`, které již nemusí být podporováno.

6.4 BitmapPreProcessor

Funkcí této třídy je převod vstupního barevného obrázku do černobílého. Kvůli třídě `BlobCounter` byla zvolena varianta, kde je pozadí černé a zájmové objekty bílé.

```
class BitmapPreProcessor : IPreProcessor<Bitmap>
```

Jedná se o složení několika filtrů z knihovny `AForge` do jednoho „filtru“. Těmito filtry jsou již zmíněné (viz sekce 4.3) inverze barev, převod na stupně šedi a prahování. Tyto filtry jsou na bitmapu aplikovány přesně v tomto pořadí.

Primární důvod pro vytvoření tohoto rozhraní bylo větší odstínění knihovny `AForge`. Díky tomu můžou být budoucí rozšíření nezávislá na knihovně `AForge`. `BitmapPreProcessor` mohl též implementovat rozhraní `IFilter` ze jmenného prostoru `AForge.Imaging.Filters`, ovšem tím by bylo vynuceno použití `AForge` i v budoucích modifikacích, čemuž jsem se chtěl vyhnout. Tyto modifikace by mohly spočívat třeba ve změně přístupu k objektům. Aktuálně detekujeme objekty pouze podle barvy, ale algoritmy na detekci pohybu pracují se změnou pozadí, a na barvě jako takové jim nezáleží.

6.5 CurveMask

Tato abstraktní třída reprezentuje masku, která má definovanou uzavřenou křivku. Body se poté dělí na ty uvnitř křivky (`IsIn` vrací `true`) a mimo křivku (`false`).

```
abstract class CurveMask : IMask<AForge.Point>
```

Od této třídy jsou odvozené dvě další třídy `RectMask` pro obdélníkové masky a `EllipMask` pro masky elipsovitého tvaru.

```
class RectMask : CurveMask
class EllipMask : CurveMask
```

6.6 PictureBlobCounter

`PictureBlobCounter` vezme černobílý obrázek a vrátí dvojice shluků spolu s označením masky, do které shluk patří.

```
class PictureBlobCounter : ICounter<Bitmap, Blob, AForge.Point>
```

Metoda `CountItems` vezme bitmapový obrázek ze vstupu a najde všechny shluky, které splňují podmínku velikosti. Tato podmínka závisí na výběru uživatele, zda chce filtrovat dle plochy shluku v pixelech, nebo dle rozměrů shluku (výška, šířka). Minimální i maximální mezní hodnota je měřena vždy v počtech pixelů a je nastavitelná v levé části okna.

```
List<Tuple<string, Blob>> output = new List<Tuple<string,
↪  Blob>>();

foreach (var blob in blobCounter.GetObjectsInformation())
    foreach (var mask in Masks)
        if (mask.IsIn(blob.CenterOfGravity) && !output.Any(tpl =>
        ↪  tpl.Item1 == mask.Tag && tpl.Item2 == blob))
            output.Add(new Tuple<string, Blob>(mask.Tag, blob));

return output;
```

Samotný výběr a párování shluků k maskám probíhá v jednom `foreach` bloku. Každý `blob` je otestován, jestli jeho těžiště náleží dané masce (`IsIn`). Pokud dojde ke shodě je `blob` otestován, jestli již nebyl jednou přiřazen jiné masce stejného označení. Toto testování je důležité kvůli skládání masek. Pokud uživatel chce provést maskování složitějšího tvaru, než je pouze obdélník nebo elipsa, tak je potřeba větší masku poskládat z menších. Proto ve chvíli, kdy mají dvě masky stejné označení, budou chápány jako jedna oblast. Shluk, který by se nacházel ve dvou překrývajících se maskách, bude započítán pouze jednou. Naopak překryv dvou masek různého označení povede ke dvojímu započítání shluku.

Na výstup jsou vydávány dvojice `označení masky, shluk`. Toto označení je potřeba kvůli následnému přiřazení shluků k příslušným instancím rozhraní `IKeeper`. Tímto je zaručeno, že `IKeeper` dostane již pouze validní shluky a tyto shluky není potřeba nadále znovu filtrovat. Veškeré filtrování by samozřejmě mohlo probíhat i mimo třídu `PictureBlobCounter`, ale takto probíhá veškeré filtrování na jednom místě — jak poziční, tak velikostní. Metoda vrátí validní shluky, o kterých má smysl uvažovat jako o objektech.

6.7 BlobData

Tato třída udržuje kolekci instancí třídy `Blob`. Na měření podobnosti shluku s trasou i na měření vzdálenosti jsou používánna čísla s plovoucí desetinnou čárkou (`double`). Pozice je vyjádřena jako bod ve dvoudimenzionálním prostoru (`AForge.Point`).

```
class BlobData : IData<Blob, double, double, AForge.Point>
```

Do této třídy je možné vkládat jednotlivé shluky a třída sama udržuje přehled o tom, jak se trasa objektu vyvíjí. Udržované údaje jsou okamžitá rychlost, která je měřena jako rozdíl těžiště dvou posledních shluků, a poté průměrovaná rychlost, což je průměr okamžitých rychlostí napříč celou historií. Dalšími udržovanými údaji jsou okamžitá a průměrná velikost, obojí měřeno v počtech pixelů. Jednotlivé velikosti jsou získávány rovnou z třídy `Blob`.

Směr objektu je normalizovaný rozdíl vektorů dvou posledních pozic.

```
(First.CenterOfGravity - Second.CenterOfGravity).Normalize()
```

Ze směru, rychlosti a pozice posledního přidaného objektu jsme schopni predikovat, kde se bude sledovaný objekt nacházet na dalším snímku.

```
averageSpeed * Direction + First.CenterOfGravity
```

Zde předpokládáme, že sledovaný objekt nebude náhle a nepředvídatelně měnit směr pohybu třeba, že se neotočí o 180 stupňů a nebude pokračovat v pohybu dozadu. Na tomto principu funguje tzv. Kalmanův filtr (viz Kalman (1960) a Grewal (2011)), který na základě aktuálního stavu a předchozího pozorování systému umí vcelku přesně odhadnout budoucí stav systému.

Funkce, která měří míru podobnosti mezi shlukem pixelů (`Blob`) a danou trasou, je euklidovská vzdálenost těžiště shluku a predikce pozice.

6.8 BlobKeeper

Tato třída slouží k udržování několika tras, které mají něco společného. V našem případě je to maska, do které dané trasy náleží.

```
class BlobKeeper : IKeeper<Blob, double, double, AForge.Point>
```

Jedinou funkcí této třídy je přiřadit jednotlivým trasám shluky pixelů získané předchozím výpočtem. K tomu využívá porovnávací funkci tras (viz kapitola 6.7). Naším cílem je co nejpřesněji přiřadit jednotlivé shluky na trasy, tak aby docházelo k co nejmenším chybám. Takto chceme minimalizovat cenu jednotlivých přiřazení. S tím se pojí několik problémů. Primárně je potřeba nespoléhat se na to, že na každém snímku dostaneme stejný počet objektů jako máme tras. Jednotlivé objekty se mohou slévat dohromady, mohou se rozpojovat již dříve slité a nebo mohou vystoupit či vstoupit zpoza hranice masky.

Nejjednodušší párovací metoda, je přiřadit každý objekt té trase, která je mu nejblíže. Tato metoda funguje do chvíle, kdy jsou od sebe objekty dostatečně vzdálené. Ale ve chvíli, kdy by se nám měly dvě trasy potkat, dojde ke splynutí tras, protože nehlídáme abychom přiřazovali každému objektu právě jednu trasu.

Složitější varianta je založena na *assignment task problem* (viz Kuhn, 1955). V tomto problému máme zadáno $n \in \mathbb{N}$ řekněme řemeslníků, n prací a matici cen, kde na pozici $[i,j]$ je cena za jakou řemeslník i vykoná práci j. Naším cílem je vybrat pro každého řemeslníka právě jednu práci, tak aby celková cena provedení všech prací byla co nejmenší.

Náš problém je stejný, jen, díky různému počtu tras a shluků, nemáme zaručenou čtvercovou matici na vstupu. Z tohoto důvodu je zavedena penalizační

hodnota, která udává o kolik pixelů se sledovaný objekt určitě mezi snímky neposune. Tyto penalizační hodnoty slouží ve finální matici jako výplň, která nám zaručí, že se každá trasa spáruje na něco a stejně tak objekty.

Takže pro n tras a m objektů vytvoříme čtvercovou matici $M \in \mathbb{R}^{2(max(m,n)}$. Ve chvíli kdy máme perfektní přiřazení (tj. přiřazení s co nejmenší cenou), tak musíme provést filtraci výsledků.

Trasy, které se spárovaly s validními objekty, jsou v pořádku, pouze zavoláme funkci `AddItem`, která aktualizuje trasu. Penalty, které se spárovaly s penaltami, nás nezajímají vůbec. Objekty, které nemají pár v trasách, jsou ty objekty, které pravděpodobně vznikly mezi posledními dvěmy snímky — může jít o nový objekt na ploše, vlivem dopadu světla do nynějška nebyl objekt vidět apod. Proto jim musíme vytvořit novou trasu.

Poslední co nám zbývá jsou trasy, které nenalezly pokračování v objektech. U těchto tras mohlo dojít k útěku objektu ze záběru, či ke splynutí dvou objektů do jednoho shluku. U těchto tras dojde k poznačení. Pokud není této trase do k snímků přiřazen shluk, je definitivně odstraněna. V opačném případě je trasa opět revalidována.

Volba správné penalty a čísla k je důležité. Při volbě malé penalty může docházet k opouštění validních tras a tím pádem i velkému nárůstu nových tras. Naopak příliš velká penalta může vést k nesprávnému mapování objektů. Ovšem pro správný běh programu je lepší používat spíše větší penalty, neboť vykazují menší chybovost v při trasování.

6.9 Matrix

Problémem perfektního přiřazení se zabývá třída `Matrix`. Tato dostává na vstupu dvě enumerace a distanční funkci. Vytvoří si matici $M \in \mathbb{R}^{2(max(m,n))}$, kde m je počet prvků první enumerace, n je počet prvků té druhé. Nyní je celá matice vyplněna buď vzdáleností dvou příslušných prvků, nebo penalizační konstantou. Distanční funkce se použije v podmatici $M \in \mathbb{R}'^{m \times n}$, kde se nacházejí vzdálenosti prvků ze zadaných enumerací. Na ostatních indexech je nastavena penalta.

Ve chvíli, kdy je vstupní matice připravená, je spuštěn algoritmus na hledání nejmenšího přiřazení. Použitý algoritmus se nazývá `Hungarian algorithm` podle země původu tohoto algoritmu (viz Kuhn, 1955). Po nalezení ideálního – tedy nejlevnějšího přiřazení – jsou výsledky rozděleny na tři skupiny. Tyto skupiny jsou poté vydány na výstup volání funkce `GetPerfectAssignment`.

- Pravé páry – došlo k napárování prvku jedné enumerace k prvku enumerace druhé,
- Nespárované „úkoly“ – došlo k napárování některého úkolu (*Task*) na penalizujícího agenta,
- Nespárovaní „agenti“ – došlo k napárování některého agenta (*Agent*) na penalizující úkol.

6.10 AppendableStream & AppendableStreams

Tyto třídy slouží ke korektnímu výpisu všech údajů. Každá maska je reprezentována právě jednou instancí rozhraní `IKeeper`, v rámci něhož mohou vznikat a zanikat jednotlivé trasy (tj. i sloupečky výstupu). Kdybychom vypisovali všechny instance `IKeeper` do jednoho souboru, vedlo by toto chování ke spřeházení dat. Nově vzniklá moucha na první masce nám posune všechny výstupy o počet sloupečků potřebných k vypsání informací o jedné mouše. Proto má každá maska přiřazen jednu instanci třídy `StreamWriter`. Tyto instance jsou poté v rámci přehlednosti kódu sdruženy do jedné instance třídy `AppendableStreams`, která se stará o správné přiřazení správných instancí `StreamWriter` ke správným instancím `IKeeper`.

Z důvodu dynamického objevování a ztrácení objektů je problematický i výstup hlavičky finální tabulky, neboť samotná podoba hlavičky je známá až po skončení analýzy. Proto jsou výstupy po celou dobu běhu analýzy vypysovány do dočasného souboru, odkud jsou po skončení analýzy – a tedy i vytištění hlavičky – přepsány do finální podoby tabulky.

Jednodušší možností implementace by bylo poněkud nepohodlné vypisování hlavičky za tabulku. Toto řešení by znesnadňovalo uživateli následné zpracování tabulky, protože většina statistických programů (`R`, `MS Excel`, atd.) očekává tabulku před daty.

7. Návod k použití

Tato kapitola slouží jako uživatelský manuál. Je zde vysvětleno, jak zacházet s programem `FlyCatcher`, tak aby byla výsledná analýza co nejpřesnější.

7.1 Instalace

Program `FlyCatcher` je vyvíjený v prostředí `.NET 4.6` a pro jeho spuštění je potřeba mít připravené dynamicky linkované knihovny.

- `AForge.dll`
- `AForge.Imaging.dll`
- `AForge.Math.dll`
- `AForge.Video.dll`
- `AForge.Video.FFMPEG.dll`
- `AForge.Video.VFW.dll`

Ke správnému běhu aplikace stačí mít nainstalované prostředí `.NET 4.6` a příslušné `.dll` knihovny ve stejné složce, z jaké je program spuštěn.

Navíc je připraven instalátor (viz příloha Program), který zajistí případnou instalaci prostředí a nakopírování knihoven do správné složky.

7.2 Běh programu

Program `FlyCatcher` ve zpracovávaném videu lokalizuje zájmové objekty a následně vypíše do `.csv` tabulky zjištěné informace.

Po spuštění se program nachází v iniciační fázi (viz obr. 7.2). Kdy je potřeba nahrát analyzované video. Poté má uživatel možnost vyladit parametry analýzy. Poté co byla spuštěna analýza není možné tyto parametry měnit.

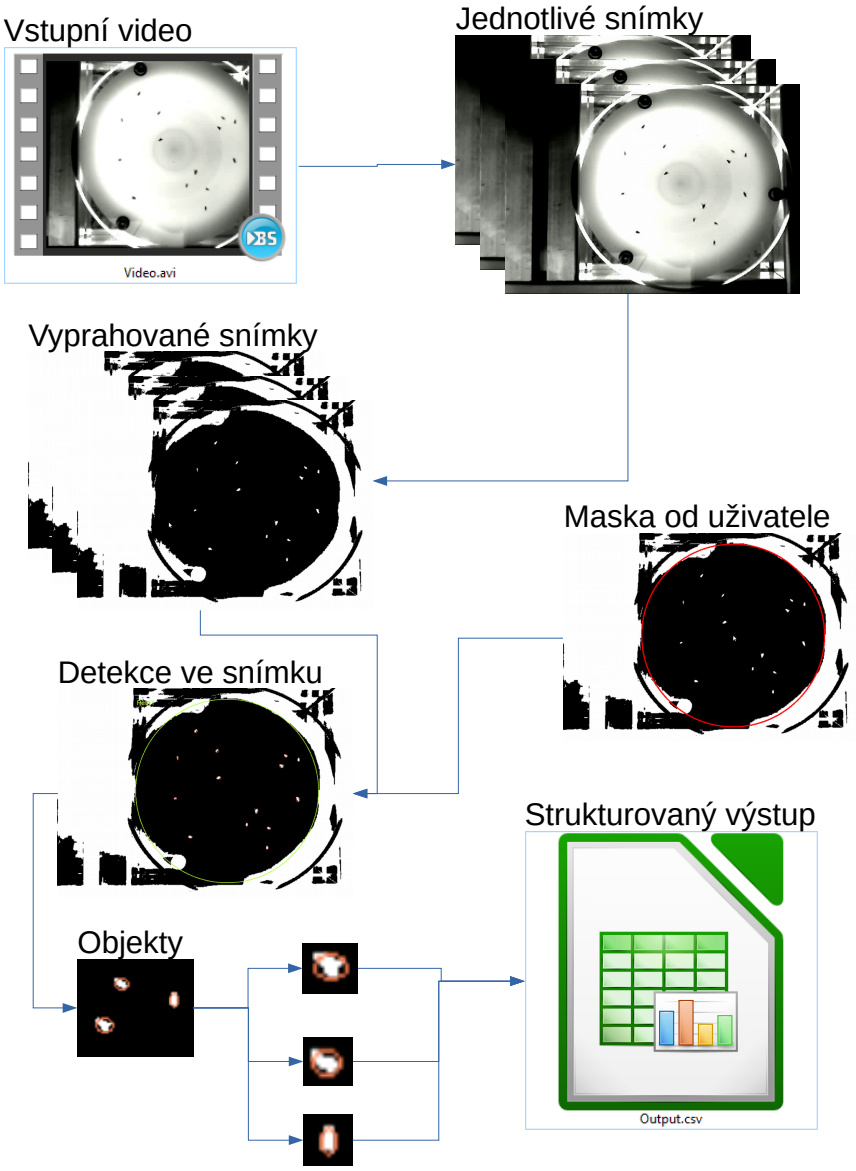
7.3 Výběr videa

Video ke zpracování lze vybrat dvěmi způsoby. Prvním je otevření souboru přes dialogové okno a druhým je přetažení daného souboru do okna aplikace – tzv. `Drag&Drop`.

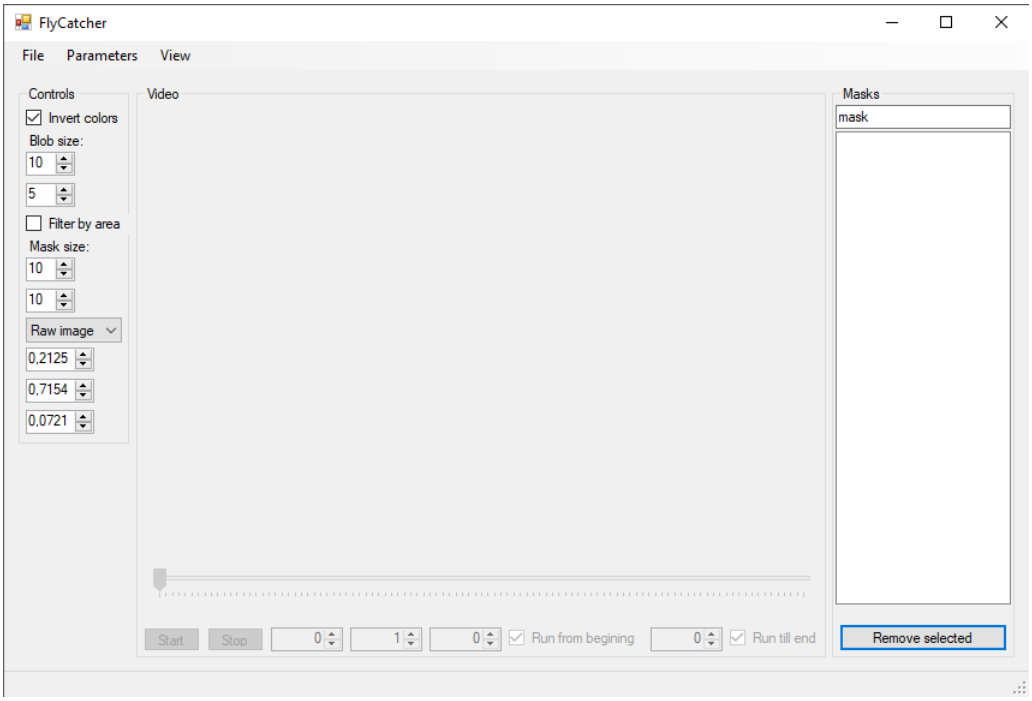
Aktuálně podporovaným formátem videa je `.mpeg`, kde je ovšem omezena schopnost posouvání po videu. Dalším podporovaným formátem je `.avi`, který ovšem může mít problémy s kodeky. Otestovaný `.avi` formát je `divx`.

Speciálním druhem zpracování videa je zpracovávání po jednotlivých obrázkových souborech. Jedná se o experimentální implementaci, která by se v budoucnu měla vylepšit na online zpracování videií rovnou při pořizování snímků kamerou přes komunikaci `FlyCatcher` ↔ `Raspberry Pi` s kamerou.

Při zpracování jednotlivých snímků je kladen důraz na správný formát názvu snímků. Očekávaný formát je <`jméno`>`číslo.koncovka`. `Číslo` udává pořadové číslo snímku a je potřeba dodržet i fixní počet cifer (tj. použít 001 místo 1).



Obrázek 7.1: *Diagram zpracování multimediálního vstupu*



Obrázek 7.2: *Program* `FlyCatcher` *po spuštění*

Dále je důležité předávat vždy pouze poslední snímek, který chceme zpracovávat. Pořadové číslo tohoto snímku totiž určuje horní zarážku na zpracování. Další předpoklad je, že zpracujeme všechny snímky od 0 až do snímku daného na vstupu. Tedy ve chvíli předání souboru `Mouchy50.jpg` začne analýza od snímku `Mouchy00.jpg`, následuje snímek `Mouchy01.jpg`, a tak dále až ke snímku `Mouchy50.jpg`.

Pokud by uživatel přidal více souborů najednou, tak by došlo k přepsání načtených dat, což může vést k neočekávanému chování – například se analýza zastaví dříve, než by měla.

7.4 Ovládání programu

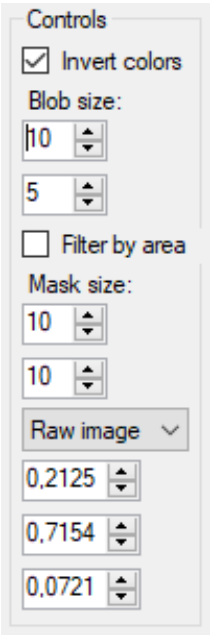
Nastavení paramterů může probíhat dvěmi způsoby. Buď ručně od uživatele přímo v okně aplikace, nebo přes nahrání konfiguračního souboru. Pro nahrávání externího konfiguračního souboru byla implementována podpora pro výber souboru pomocí dialogového okna a nebo pomocí `Drag&Drop`.

Po spuštění programu se uživatel nachází v iniciační fázi. Nyní je potřeba nastavit veškeré parametry:

- Velikost shluků - určuje jak velké a jak konkrétně se mají shluky filtrovat,
- Obrazový filtr - předzpracování obrazu,
- Masky - definování zájmových oblastí analýzy,
- Rozsah analýzy - v jaké části videa má analýza probíhat.

7.4.1 Controls

V sekci `Controls` se nachází kontrolky pro globální nastavení zpracování videa. Samotné vyplnění hodnot je pro uživatele systému `Windows` běžné, proto rozebereme pouze dopady na samotnou analýzu:



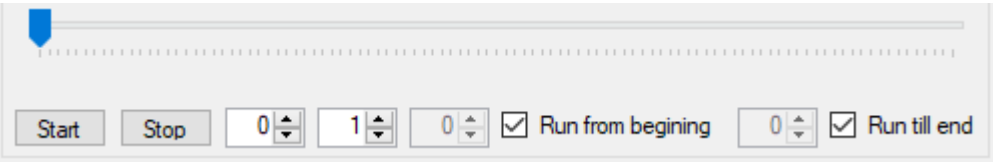
Obrázek 7.3: *`Controls`*

- `checkBox Invert colors` invertuje barvy v obrazu,
- První dva číselníky shora nastavují maximální (resp. minimální) velikost shluků,
- Další `checkBox` rozhoduje, zda-li se mají shluky filtrovat podle rozměrů nebo podle počtu pixelů,
- Následující dva číselníky určují velikost zadávané masky v x-ovém a y-ovém směru,
- Nabídka výběru přepíná mezi zobrazením originálního snímku (`Raw image`),vyprahovaného (`Processed`) a žádného (`None`),
- Poslední tři číselníky nastavují `RGB` váhy jednotlivých barev u černobílého filtru.

7.4.2 Video

Pod snímkem se nachází ovládací prvky samotného videa.

- Klasický posuvník výběru aktuální pozice videa,
- Dvě tlačítka pro zastavení a spuštění analýzy,
- Číselník ukazujícíc aktuální číslo snímku,
- Další číselník určuje velikost kroku,
- Poté následují dva číselníky a dva přepínače, číselník určuje mezní hodnoty běhu videa a přepínače, zda se na tyto hodnoty má brát zřetel.

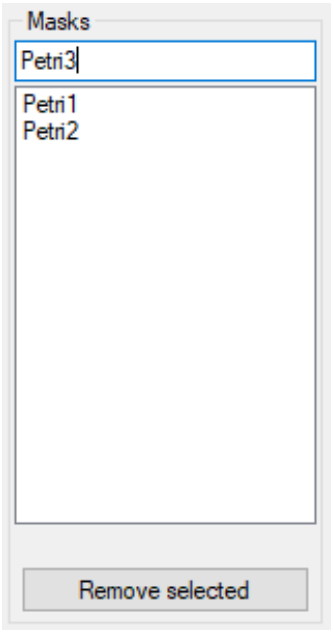


Obrázek 7.4: *Kontrola toku videa*

7.4.3 Masks

V pravé části aplikace se nachází seznam masek.

- Textové pole určuje aktuální `Tag` masky,
- Následuje seznam všech aplikovaných masek,
- Tlačítko ve spodní části slouží k odstranění nechtěných masek.



Obrázek 7.5: *Masks*

7.5 Funkce parametrů

Každý parametr má svůj specifický vliv na zpracování videa a pro správný běh programu je potřeba tyto parametry nastavit správně a vědět co přesně dělají.

7.5.1 Zpracování obrazu

Programová komponenta hledající shluky pixelů předpokládá, že zájmové objekty jsou světlé a pozadí je naopak černé. Pokud by byly zájmové objekty naopak tmavé, je potřeba invertovat barvy zpracovávaného videa. Tuto funkcionalitu zajišťuje právě přepínač `Invert colors`.

Pro správnou detekci zájmových útavrů, je důležité mít nastavenou správně prahovací funkci. K tomu slouží parametry pro barevné váhy. Každý obrázek je nejprve převeden do černobílého obrázku, kde barva každého pixelu je brána jako vážený průměr jeho tří barevných složek (7.1).

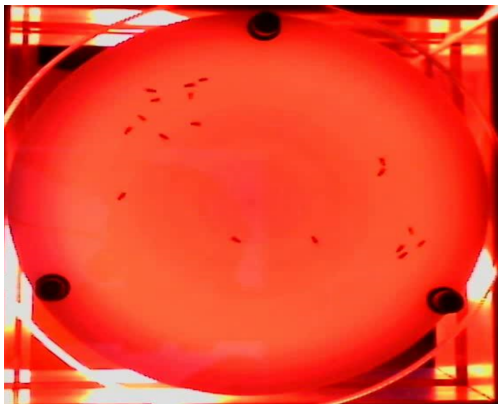
$$\begin{aligned} \texttt{šedá} = {} & \texttt{váha červené} * \texttt{červená složka} + \\ & + \texttt{váha zelené} * \texttt{zelená složka} + \\ & + \texttt{váha modré} * \texttt{modrá složka} \end{aligned} \tag{7.1}$$

Z obrázku 7.6 je patrné, že na správném nastavení parametrů záleží. Na obrázku 7.6b jsou všechny zájmové objekty dobře patrné ovšem ztratili jsme jistou část pozorovací plochy, která svojí barvou splývala s mouchami. Obrázek 7.6c prozměnu zachovává daleko větší část z pozorovací plochy, ovšem některé mouchy jsou po vyprahování velmi špatně rozpoznatelné. Proto je důležité mít na paměti, že strojové zpracování není dokonalé a ani nejlepší vyladění parametrů nemusí být dokonalé.

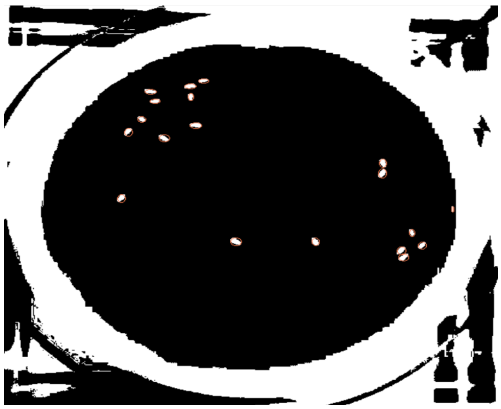
7.5.2 Velikost shluků

Primární filtrování shluků probíhá podle velikosti. Číselníky `Blob size` nastavují mezní hodnoty této filtrace. Na základě nastavení přepínače `Filter by area` se mění význam těchto velikostí.

Pokud filtrujeme podle počtu pixelů (tj. `Filter by area` je zaškrtnut), určují číselníky maximální a minimální plochu shluku (měřeno v počtu pixelů). Ve chvíli, kdy políčko `Filter by area` není zakšrtnuto, tak probíhá filtrace podle délky v x-ovém nebo y-ovém směru.



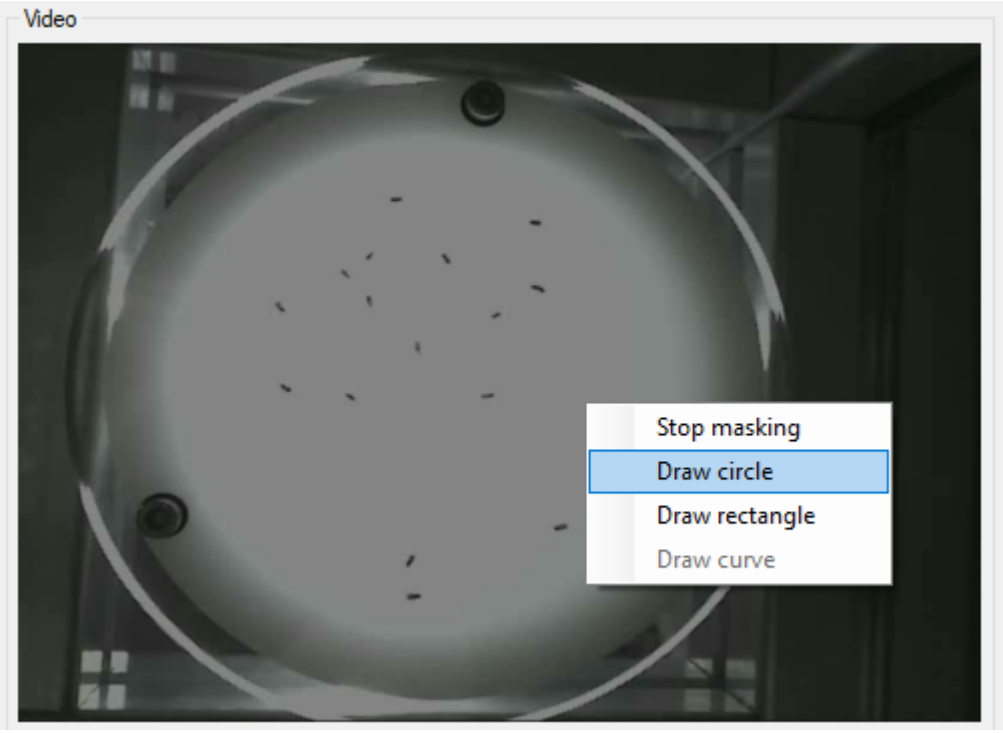
(a) *Originál*



(b) *Vyladěné váhy*

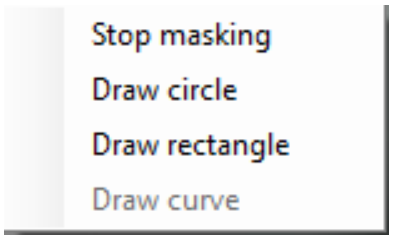
(c) *Přednastavené váhy*

Obrázek 7.6: *Porovnání tří obrázků s různým nastavením* `RGB` *vah*



Obrázek 7.7: *Počátek maskování*

7.5.3 Masky



Obrázek 7.8: *Kontextová nabídka*

Důležitou funkcionalitou je vybrání správných zájmových oblastí. K tomu slouží tzv. masky. Tyto se dají použít i k odlišení jednotlivých oblastí zájmu v jednom videu. Maska poté rozlišuje shluky pixelů na vnitřní a vnější, které ignoruje.

Masku vytvoříme buď nahráním konfiguračního souboru (viz kapitola 7.6) nebo přímo v okně samotné aplikace. V okně aplikace je potřeba kliknout pravým tlačítkem myši do oblasti, kde se nachází snímek z nahraného videa. Tím se zobrazí kontextová nabídka masek (viz obr. 7.8).

Aktuálně jsou implementovány dva základní tvary masek. Prvním je elipsa a druhým tvarem je obdélník. Ve chvíli, kdy si uživatel vybere tvar masky (`Draw circle` nebo `Draw rectangle`) se aplikace dostane do kreslícího stavu. Pozice kurzoru ve snímku označuje střed aktuálně přidávané masky. Kliknutím do snímku dojde k zafixování masky a přidání mezi aktivně používané masky. To se projeví zvýrazněním zjištěných objektů a objevením masky v nabídce masek (na obr. 7.5 viz „`Petri1`“ a „`Petri2`“). Již přidané masky nelze dále upravovat.

Název přidávané masky (`Tag`) určuje o jakou zájmovou oblast se jedná. Díky tomu lze poskládat i složitější tvary než je obdélník nebo elipsa. Ovšem vyžaduje to přidání několika masek se stejným jménem. Tyto jsou poté množinově sjednoceny a uživatel může pokrýt i oblasti, které by nešly pokrýt obdélníkem nebo elipsou.

Nalezené objekty jsou zpracovány každou různou maskou, do které náleží. Takže ve chvíli, kdy se překrývají dvě masky se stejným názvem jsou objekty v jejich průniku zaznamenány pouze jednou. Pokud se ovšem překrývají dvě masky s různými jmény dojde k započítání objektu vícekrát.

Na obrázku 7.9 byla křivě položená testovací trubička pokryta dvěmi obdélníky, aby došlo k co nejpřesnějšímu pokrytí, kde nebude zachycen šum v podobě okraje.

Obrázek 7.9: *Průnik masek*

7.5.4 Výstupní soubor `.csv`

Celý program umožňuje ukládat výstup na disk do souboru s příponou `.csv`. Tento soubor je ovšem potřeba programu předat. To uděláme stejně jako předávání multimediálních souborů, pouze vybereme soubor s příponou `.csv`.

Každá maska odpovídá jednomu výstupnímu souboru. Tento soubor se nachází ve stejné složce, kde se nachází výstupní soubor poskytnutý uživatelem. Ve výstupním souboru se nachází seznam masek — tedy i seznam všech výstupních souborů.

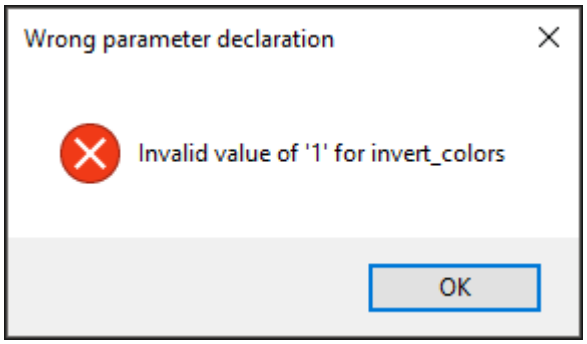
Samotný výstup trasování se nachází v souborech přidružených maskám. V prvním sloupečku se nachází počet zpracovaných snímků a v dalších poté hodnoty, které si uživatel na výstup přál.

7.6 Konfigurační soubor

Mezi konfigurační soubory jsou zařazeny soubory s koncovkou `.config`, `.mask` a `.output`. U všech konfiguračních souborů je očekáván stejný formát.

```
parametr = hodnota parametru
```

Řádky, které neobsahují „=“ nebo neobsahují validní název parametru, jsou ignorovány. Pokud by řádek obsahoval „=“ a jednalo by se o validní název parametru, ale byla by k němu přiřazena invalidní hodnota, dojde k běhové chybě a hodnota parametru bude nastavena na výchozí hodnotu. Poté dojde ke zpracování zbytku konfiguračního souboru.



Obrázek 7.10: *Chyba v konfiguračním souboru*

Následující konfigurační soubor tedy nastaví hodnotu parametru `history` na hodnotu 10, poté dojde k zobrazení chybové hlášky 7.10 uživateli a nastavení `invert_colors` na `true`, což je výchozí hodnota. Zbytek souboru je již bez chyb, takže dojde ke zpracování obvyklým způsobem.

```
history = 10
invert_colors = 1
penalty = 10
```

7.6.1 `.mask`

Název parametru	Formát	Význam	Výchozí hodnota
`clean`	`boolean`	`True` pokud mají nové masky přepsat ty staré.	`false`
`ellipse`	`mask`	Definuje elipsovitou masku.	—
`rectangle`	`mask`	Definuje obdélníkovou masku.	—

Tabulka 7.1: Jména a význam parametrů souboru `.mask`

Konfigurační soubor s příponou `.mask` slouží k uchovávání a nahrávání masek do aplikace (viz tab. 7.1). Masky jsou uloženy jako relativní souřadnice ohraničujícího obdélníku. To znamená, že `x1` označuje procentuální pozici, měřeno vůči šíři snímku, jedné strany v x-ové souřadnici. Je důležité dodržet to, že se jedná

o procenta, proto musí být všechny hodnoty z intervalu $[0, 100]$. Nedodržení toho intervalu nezpůsobí běhovou chybu, ale vytečení masky mimo zobrazovaný snímek. Toto může někdy být žádoucí chování, proto nebyla tato „chyba“ ošetřena.

```
[ellipse|rectangle] = tag x1 y1 x2 y2
```

7.6.2 `.output`

Název parametru	Formát	Význam	Výchozí hodnota
`objects`	`boolean`	Zapne výpis počtu objektů.	`true`
`avg_speed`	`boolean`	Zapne výpis průměru rychlosti.	`true`
`imm_speed`	`boolean`	Zapne výpis aktuální rychlosti.	`true`
`glb_speed`	`boolean`	Zapne výpis součtu okamžité rychlosti.	`true`
`glb_avg_speed`	`boolean`	Zapne výpis průměru okamžité rychlosti.	`true`
`position`	`boolean`	Zapne výpis aktuální pozice.	`true`
`prediction`	`boolean`	Zapne výpis pozice predikce.	`true`
`avg_area`	`boolean`	Zapne výpis průměrné plochy.	`true`
`imm_area`	`boolean`	Zapne výpis aktuální plochy.	`true`
`glb_area`	`boolean`	Zapne výpis součtu aktuálních ploch.	`true`
`glb_avg_area`	`boolean`	Zapne výpis průměru aktuální plochy.	`true`

Tabulka 7.2: Jména a význam parametrů souboru `.output`

Konfigurační soubor `.output` slouží k nastavování `.csv` výstupů. Určuje, které informace o zjištěných objektech se mají vypisovat na výstup a které naopak ne (viz tab. 7.2). Každý záznam ve finálním výstupu odpovídá jednomu snímku analyzovaného videa.

7.6.3 `.config`

Soubor `.config` slouží k všeobecnému nastavení celého programu. Mohou se v něm nacházet jak definice masek (7.6.1), tak definici výstupů (7.6.2). Navíc se v něm mohou vyskytovat další parametry nastavující celý systém (viz tab. 7.3).

Uživatel by si měl dávat pozor na postupnou aplikaci více konfiguračních souborů. Pokud totiž není v souboru nalezen parametr, tak se automaticky předpokládá jeho nastavení na výchozí hodnotu.

Název parametru	Formát	Význam	Výchozí hodnota
`history`	`integer`	Udává počet shluků, které si pamatuje trasa.	10
`penalty`	`integer`	Počet pixelů, o které se určitě nezmění poloha objektu mezi snímky.	10
`max_size`	`integer`	Horní mezní hodnota na měření velikosti shluků.	10
`min_size`	`integer`	Dolní mezní hodnota na měření velikosti shluků.	5
`height`	`integer`	Výška masky.	10
`width`	`integer`	Šířka masky.	10
`invert_colors`	`boolean`	Přepínač invertování barev.	10
`valid_threshold`	`byte`	Počet snímků než bude špatná trasa zahozena.	5
`red_coef`	`decimal`	Určuje váhu červené složky v převodu na černobílou.	0,2125
`green_coef`	`decimal`	Určuje váhu zelené složky v převodu na černobílou.	0,7154
`blue_coef`	`decimal`	Určuje váhu modré složky v převodu na černobílou.	0,0721
`mark_objects`	`boolean`	Zapne zvýrazňování objektů.	`true`
`mark_prediction`	`boolean`	Zapne zvýrazňování pozice predikce.	`false`
`mark_trace`	`boolean`	Zapne zvýrazňování předešlé trasy.	`false`
`mark_tag`	`boolean`	Zapne zobrazování „jména“ jedince.	`false`
`mark_direction`	`boolean`	Zapne zvýrazňování směru jedince.	`false`

Tabulka 7.3: Jména a význam parametrů souboru `.config`

8. Výsledky

Zde si projdeme ostatní programy, které lze použít ke sledování objektů. U TriKinetics se jedná o celou aparaturu se specifickým HW a k tomu příslušným SW. U ostatích jde pouze o samostatný SW zpracovávající video.

- TriKinetics - HW a SW pro sledování much,
- PySolo - multiplatformní software pro analýzu spánkové a pohybové aktivity much,
- CTRAX - SW řešení přímo pro sledování much,
- Tracker - SW pro sledování much v reálném čase,
- Drosana - SW řešení vytvořené přímo pro entomologickou laboratoř.

8.1 TriKinetics

HW od TriKinetics je navržený pro měření pohybové aktivity jedinců druhu *Drosophila*. Ovšem HW zařízení je dosti limitovaný. Aktivita jedince se měří pomocí počítání průletů měřeným úsekem. Měření je prováděno pomocí infračervených paprsků a je měřeno kolikrát moucha protla jaký paprsek. Detailněji pospané metody jsou v manuálu k aparatuře Tri (2007).

Obrázek 8.1: *Testovací trubička v aparatuře*

Přestože přiložené vzorce a knihovny na zpracování výstupů z aparatury od firmy TriKinetics jsou velmi precizní, nemohou postihnout veškerá chování, které může sledovaná moucha vykazovat. Moucha se může třeba pohybovat pouze mezi dvěma snímači takovým způsobem, že jí ani jeden nezachytí. Dále pak není možné spolehlivě sledovat více much v jednom prostředí, protože snímač není schopen zachytit dvě mouchy blízko sebe jako dva záznamy.

8.2 PySolo

V tuto chvíli se nejaktuálnější informace nacházejí v repozitáři na stránce `github`. Internetové stránky jsou většinu času nedostupné.

Jednou se mi povedlo se na stránky projektu dostat a stáhnout instalátor. Tento je ovšem nefunkční, alespoň tedy v prostředí `Windows 10`, kde jsem jej zkoušel.

8.3 CTRAX

Přestože je `CTRAX opensource` program, je jeho použití úzce vázáno na komerční program `Matlab`. Výstup z analýzy videa je uložen ve formátu `.ann`, který je potřeba zpracovávat právě pomocí připravených maker v programu `MATLAB`.

Obrázek 8.2: *Okno aplikace* ***CTRAX***

Detekce much pracuje na základě metody odečítání pozadí. Nejprve se určí, jak vlastně vypadá zkoumaná plocha bez much (pozadí). Poté jsou v každém snímku zájmové objekty detekovány pomocí rozdílu barvy pixelu snímku a barvy pixelu pozadí. Zájmový objekt je detekován, pokud mají všechny jeho pixely tento rozdíl větší než `low threshold` a nacházejí se „blízko“ pixelu, kde je tento rozdíl větší než `high threshold`.

Tato metoda detekce je velmi spolehlivá a funguje i s malým množstvím nastavitelných parametrů. Ovšem její použití s sebou nese jistá rizika. Velkým problémem pro CTRAX se stávají stacionární mouchy, které také chceme detekovat. Pokud se totiž moucha po celou dobu běhu videa nepohne, je automaticky klasifikována jako pozadí, a tím pádem o ní nedostáváme žádné informace. Tento problém je vyřešen manuálním označením stacionárních much na pozadí, kde dojde k interpolaci barvy a „vymazáním“ této mouchy z pozadí, čímž se dostane na výstup.

8.4 Tracker

Program Tracker je naimplementován v jazyce JAVA, díky čemuž se jedná o přenostielný SW. Ovšem použití je vázáno na systém, v jehož prostředí jde JAVA přeložit a následně teprve spustit. Dále pak je nutné pořizovat a spracovávat videa pouze v reálném čase.

Program je z internetových stránek stažen v podobě .zip archivu. Samotné použití aplikace je pro běžného počítačového uživatele dosti zmatečné a zdlouhavé. Ale přiložený návod (viz Bra, 2012) vysvětluje vše vcelku podrobně.

Hlavní nevýhodou tohoto SW je samotné uživatelské rozhraní, které neexistuje. Všechny parametry si uživatel musí nastavit přímo v kódu aplikace. Dalším nedostatkem je nemožnost sledovat více much na jedné ploše. Každá moucha musí být při experimentu uzavřena v separátním prostoru.

8.5 Drosana

Drosana je program, který byl vyvíjen přímo pro Laboratoř molekulární genetiky. HW řešení spočívá v pořizování videí, která se poté mohou zpracovat. O toto zpracování se stará program Drosana (*DROSophila ANAlyzator*). Ten je vyvíjen v jazyce Visual Basic pro prostředí .NET. Jeho výstupy jsou rovnou ukládány do programu Excel, což může v některých případech způsobit nečekané problémy. Další nevýhodou je aktviní využívání systémové schránky ke zpracování videa. Díky tomu se počítač, na kterém běží analýza, stává prakticky nepoužitelným.

8.6 *Beta testing*

Program FlyCatcher byl otestován na laboratorních videích. První test spočíval v pozorování schopnosti programu udržet trasu jednotlivých objektů (viz příloha Test trasování).

◁ Vstupní video soubor - Video.avi
◁ Konfigurační soubor - names.config a track.config

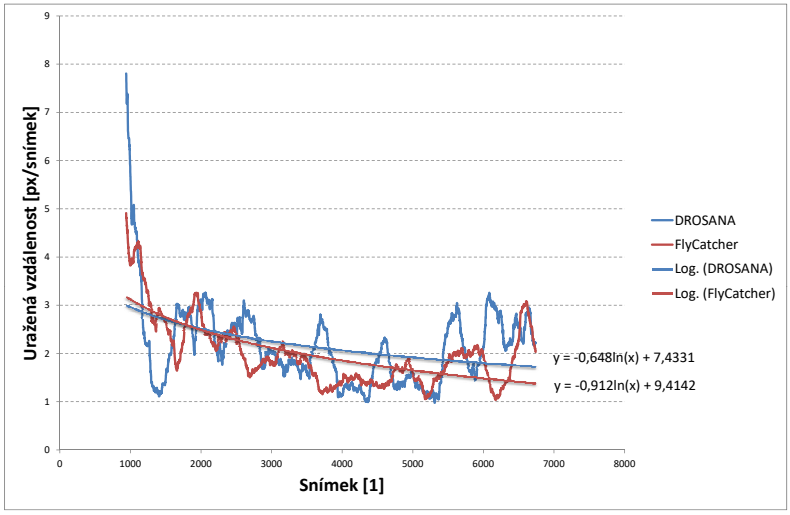
Konfigurační soubor names.config zapíná zobrazování identifikátorů, soubor track.config zobrazuje pouze samotné trasy.

V tomto testu je vidět, že program je schopen si, při správném nastavení parametrů, udržet přehled o objektech. A to i v méně přehledných situacích, jako

je spylnutí dvou objektů do jednoho shluku pixelů (snímek 100 až 140, pravý horní roh).

Dalším testem bylo porovnání výstupů z programu `Drosana` s výstupy z programu `FlyCatcher`.

Pokus na analyzovaném videu spočívá v měření aktivity much ve stresových podmínkách. Aktivita je měřena v počtu pixelů, kteří sledovaní jedinci urazí mazi dvěma snímky (viz příloha Porovnání výstupů).



Obrázek 8.3: *Porovnání výstupů programů* `FlyCatcher` *a* `Drosana`

Klouzavý průměr naměřených dat je zobrazen v grafu 8.3. Tento laboratorní pokus spočívá v porovnávání úbytku aktivity mezi několika testovacími vzorky. Aby bylo toto porovnání co nejjednodušší jsou tyto průměry proloženy logaritmickou křivou.

Na tomto grafu lze vidět, že program `FlyCatcher` vykazuje podobné výsledky jako program `Drosana`. Na logaritmických křivkách je vidět, že u obou dochází po stresové události k průběžnému útlumu aktivity. Nesrovnalosti křivek jsou způsobeny jiným návrhem aplikací a jiným přístupem k eliminaci šumu, detekci jedinců a tím pádem i k jiným výsledným hodnotám pro jednotlivé snímky. Při počátečních testech programu DROSANA byl zjištěn podobný rozdíl ve výstupních datech i při nepatrné změně vstupních parametrů analýzy (například prahovacích úrovních).

Závěr

Cílem této práce bylo vylepšit podmínky zpracování videa pro Laboratoř molekulární genetiky, Entomologický ústav, Biologické centrum AV ČR. Tato laboratoř se mimo jiné zabývá studiem chování octomilky obecné (*Drosophila melanogaster*).

Jedním z metod výzkumu je sledování pohybové aktivity zmíněné octomilky. Testované subjekty jsou umístěny na petriho misku nad kteoru je zavěšena kamera. Tato poté pořizuje videozáznam s chováním sledovaných jedinců. Získaný videozáznam je následně strojově zpracován programem `FlyCatcher`. Ten video převede na faktické údaje o poloze, rychlosti či velikosti a směru jednotlivých mušek.

`FlyCatcher` po snímcích zpracovává poskytnuté video, ve kterém detekuje zájmové objekty. V každém snímku dojde k nalezení objektů a k jejich párování na jednotlivé trasy. Tyto trasy jsou poté předmětem strukturovaného výstupu, který bude v budoucnu sloužit k analýze vzorců chování zkoumaných jedinců.

Vytvořený program umí zpracovávat videa v požadovaných formátech. Díky možnosti nevykreslovat každý snímek, může běžet analýza rychlostí 15 snímků za sekundu. To by umožňovalo následně propojit program `FlyCatcher` s kamerou na `HW` aparatuře a zpracovávat snímky ihned jak jsou pořízeny.

Trasování objektů funguje. Sice je nutné pečlivě vyladit správně parametry zpracování (viz kapitola 7 a 7.6), ale díky možnosti si tyto parametry ukládat, stačí tyto parametry pro daný pokus vyladit jednou.

Díky návrhu programu a použití vlastností jazyka `C#`je `FlyCatcher` modulární program, který je lehce rozšiřitelný o nové funkce. Do budoucna se počítá s těmito rozšířeními:

- Vylepšení párovacího algoritmu -
  aktuálně je trasování velmi citlivé na vstupní parametry, adaptivní ladění těchto parametrů za běhu, by mělo tento neduh odstranit,
- Zpracování multimediálních souborů -
  do budoucna by měla vzniknout knihovna na zpracovávání záznamů v reálném čase, která bude komunikovat přímo s kamerou `HW` aparatury, díky tomu by nebylo třeba uchovávat celá videa,
- Nové možnosti zpracování videa -
  aktivita jedinců se v laboratoři měří i na larvách, ovšem v těchto případech měřit aktivitu pomocí pozorování těžiště může být zavádějící, proto se měří pouze počet změněných pixelů mezi snímky,
- Možnost vložit do videa měřítko -
  toto měřítko by umožňovalo vydávat výstupu v *SI* jednotkách.

Seznam použité literatury

Bradley, D. a Roth, G. (2007). Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, **12**(2). URL http://people.scs.carleton.ca/~roth/iit-publications-iti/docs/gerh-50002.pdf.

(2012). *Tracker User's Guide*. Brandeis university, 2.2 edition. URL http://www.bio.brandeis.edu/tracker/Tracker/RunTrackerWebInstructionsV.2.2.pdf.

Colomb, J., Reiter, L., Blaszkiewicz, J., Wessnitzer, J. a Brembs, B. (2012). Open source tracking and analysis of adult drosophila locomotion in buridan's paradigm with and without visual targets. *PLoS ONE*.

Donelson, N., Kim, E. Z., Slawson, J. B., Vecsey, C. G., Huber, R. a Griffith, L. C. (2012). High-resolution positional tracking for long-term analysis of drosophila sleep and locomotion using the "tracker" program. *PLoS ONE*, **7**(5). doi: 10.1371/journal.pone.0037250. URL https://doi.org/10.1371/journal.pone.0037250.

Gonzalez, R. (2009). *Digital Image Processing*. Pearson Education. ISBN 9788131726952. URL https://books.google.cz/books?id=a62xQ2r_f8wC.

Grewal, M. S. (2011). *Kalman Filtering*. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2˙321. URL http://dx.doi.org/10.1007/978-3-642-04898-2_321.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, **82**(1). ISSN 0098-2202. doi: 10.1115/1.3662552. URL http://dx.doi.org/10.1115/1.3662552.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, **2**.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, **9**(1). ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

Poynton, C. A. (1998). Rehabilitation of gamma. In *Photonics West'98 Electronic Imaging*. International Society for Optics and Photonics.

Štefan Kakaš (2012). Návrh a realizace přístroje na sledování hmyzu a vyhodnocování pohybové aktivity hmyzu. Master's thesis, Jihočeská univerzita v Českých Budějovicích.

(2007). *Drosophila Activity Monitoring System User's Guide*. TriKinetics Inc, v3.0 edition.

Zhang, B., Freeman, M. R. a Waddell, S., editors (2010). *Drosophila Neurobiology: A Laboratory Manual*. Cold Spring Harbor Laboratory Press. ISBN 9780879699055. URL https://books.google.cz/books?id=7odFAQAAIAAJ.

Seznam obrázků

Přílohy

Program

V adresáři `FlyCatcher` se nachází instalační soubory programu `FlyCatcher` verze 1.0.1.

Zdrojový kód

V adresáři `Code` se v textovém formátu .cs nacházejí zdrojové kódy samotné aplikace (`Form1.cs` a `Extensions.cs`), deklarace rozhraních definovaných v kapitole 5 a nakonec také tříd rozebíraných v kapitole 6.

- `BlobKeeper.cs` - rozhraní 5.1 a implementace 6.8,
- `BlobData.cs` - rozhraní 5.5 a implementace 6.7,
- `PictureBlobCounter.cs` - rozhraní 5.4 a implementace 6.6,
- `PictureGiver.cs` - rozhraní 5.1 a implementace 6.2 a 6.3,
- `PictureProcesser.cs` - rozhraní 5.2 a implementace 6.4,
- `Mask.cs` - rozhraní 5.3 a implementace 6.5,
- `Matrix.cs` - implementace 6.9,
- `AppendableStream.cs` - implementace 6.10,
- `Form1.cs` - kód provazující třídy zmíněné v 6.1,
- `Extensions.cs` - pomocné a rozšiřující funkce.

Test trasování

Součástí této přílohy je video soubor `Video.avi`, na kterém jsou sledované mouchy umístěny na podsvícenou podložku. Dále pak dva konfigurační soubory `names.config` a `track.config`. Předáním těchto souborů programu `FlyCatcher` se můžeme ujistit, že tento program je schopen správně trasovat zájmové objekty.

Porovnání výstupů

V souboru `cmp.csv` se nacházejí data, která jsou výsledkem analýzy videa programů `FlyCatcher` a `Drosana`.

- První sloupeček označuje pořadí snímků,
- druhý označuje počet pixelů, které urazili objekty mezi dvěma snímky, tak jak je vydal program `DROSANA`,
- třetí sloupeček udává tento počet spočítaný programem `FlyCatcher`,
- čtvrtý a pátý sloupeček udává klouzavé průměry uražených vzádleností.